DOCUMENT RESUME

ED 144 613 IR 005 261

AUTHOR Hultgren, Jan; Larsson, Rolf
TITLE Pastime--A System for File Compression.
INSTITUTION Royal Inst. of Tech., Stockholm (Sweden). Library.
REPORT NO TRITA-LIB-4043
PUB DATE Apr 75
NOTE 21p.; For related documents, see IR 005 260 and IR 005 263.

EDRS PRICE MF-$0.83 HC-$1.67 Plus Postage.
DESCRIPTORS Bibliographic Citations; Computers; *Computer Storage Devices; Data Bases; *Information Processing; Information Science; *Information Storage; Libraries; *Library Automation
IDENTIFIERS 3RIP Search System

ABSTRACT

An interactive search and editing system, 3RIP, is being developed at the library of the Royal Institute of Technology in Stockholm for large files of textual and numeric data. A substantial part (on the order of 10-E9 characters) of the primary file of the search system will consist of bibliographic references from a wide range of sources. If the performance of the search system is to be satisfactory, all of the primary file, or at least a large part of it, should be stored on fairly expensive secondary storage devices with relatively short access time. In order to reduce the considerable costs associated with the storage of the primary file two different approaches might be attempted: to use cheaper storage devices with longer access time for those parts of the primary file that rarely have to be processed, or use a more efficient way of coding the primary file instead of the conventional coding method. (Author)

****
* Documents acquired by ERIC include many informal unpublished *
* materials not available from other sources. ERIC makes every effort *
* to obtain the best copy available. Nevertheless, items of marginal *
* reproducibility are often encountered and this affects the quality *
* of the microfiche and hardcopy reproductions ERIC makes available *
* via the ERIC Document Reproduction Service (EDRS). EDRS is not *
* responsible for the quality of the original document. Reproductions *
* supplied by EDRS are the best that can be made from the original. *
****

ED144613

REPORT TRITA-LIB-4043 APRIL 1975

U.S. DEPARTMENT OF HEALTH, EDUCATION & WELFARE
NATIONAL INSTITUTE OF EDUCATION

THIS DOCUMENT HAS BEEN REPRODUCED EXACTLY AS RECEIVED FROM THE PERSON OR ORGANIZATION ORIGINATING IT. POINTS OF VIEW OR OPINIONS STATED DO NOT NECESSARILY REPRESENT OFFICIAL NATIONAL INSTITUTE OF EDUCATION POSITION OR POLICY

PASTIME - A SYSTEM FOR FILE COMPRESSION

Jan Hultgren and Rolf Larsson

"PERMISSION TO REPRODUCE THIS MATERIAL HAS BEEN GRANTED BY

Stephan Schwarz

TO THE EDUCATIONAL RESOURCES INFORMATION CENTER (ERIC) AND USERS OF THE ERIC SYSTEM."

IR000 261

PASTIME: A SYSTEM FOR FILE-COMPRESSION

TABLE OF CONTENTS

Introduction 1
The problem and some definitions 2
Existing methods 3
The method of PASTIME 3
Testing PASTIME 8
The encoder and the decoder; economic considerations 14
Summary 15
References 17

ABSTRACT

PASTIME is a system which compresses large fairly homogeneous computer files of textual data by uniquely translating variable-length character strings, terms, into variable-length codes. A large number of candidate terms are chosen according to their coverage of a sample of the file to be compressed. The code-lengths are chosen for easy handling in a specific computer environment. The one-to-one mapping of the terms onto the codes is generated through an iterative procedure wherein the total length of a coded sample of the file is minimized.

This mapping is then used when the entire file is compressed.

When decoding a compressed record the mapping is realized by a decoding-dictionary and a decoder occupying in all some 150 K bits.

The cost of compressing the textual data in a large file of bibliographic references to 40 per cent of their original size is estimated to 75 Swedish crowns per million characters, while the decoding of a million characters will cost about 5 Swedish crowns. The cost estimates are based on the computer charges at Stockholms datamaskincentral.

INTRODUCTION

An interactive search and editing system, 3RIP, for large files of textual and numeric data is being develolped at the library of the Royal Institute of Technology in Stockholm (5). A substantial part (on the order of 10^9 characters) of the primary file of the search system will consist of bibliographic references from a wide range of sources. If the performance of the search system is to be satisfactory all of the primary file or at least a large part of it should be stored on fairly expensive secondary storage devices with relatively short access time. Then in order to reduce the considerable costs associated with the storage of the primary file two different approaches might be attempted: to use cheaper storage devices with longer access time for those parts of the primary file that rarely have to be processed, or instead of the conventional coding method use a more efficient way of coding the primary file. In the 3RIP project we have devoted some time and effort to an investigation of the second approach, which does not necessitate a continuous monitoring of the primary file in order to distribute it in some optimal manner, over different types of secondary storage.

THE PROBLEM AND SOME DEFINITIONS

In the following we will denote by

> message: an item of information written in a natural language (e.g. a title in a bibliographic reference);
>
> term: a string of characters from a natural language alphabet (which comprises the letters, ciphers, and punctuation marks) to be represented by one code;
>
> code: a bitstring used to represent a term;
>
> conventional coding method: representing one character terms by 8-bit codes.

Let the conventional coding method require c bits to encode a given set of messages and let another method require m bits. We will then say that the other method's

> reduction is 100(c-m)/c per cent;
>
> result is m/c

Our aim is to design a coding method with a reduction of at least 60 per cent when applied to bibliographic references and moreover such that it can be economically run within the 3RIP system. The latter requirement implies that the cost of encoding must be more than compensated for by the reduced storage costs and that the decoding (the returning of the messages) must be fast, cheap and in the user's interactions with the system transparent to them. It should be noted that the reduction concept, as defined here, does not consider the computer costs associated with the encoding and decoding; thus the two requirements on the coding method may well be in conflict.

The conventional coding method is efficient if the natural language alphabet has 256 characters and if these are independent and occur with equal probabilities in the messages. However, the alphabets of natural languages such as English, French, and Russian do not qualify on either of these counts. The characters of the alphabets used for the basically phonemic transcriptions of these languages certainly do not number 256

and they are interdependent and have different probabilities of occurring in so far as certain collocations of phonemes are more frequent than others.

The fact that the characters are interdependent seems to indicate that variable-length terms should be used, while the unequal probabilities of occurrence may make variable-length codes better than fixed-length codes.

EXISTING METHODS

A variety of methods aiming at compact coding of textual information have recently been discussed in the literature. Thus Heaps (1 and 2) and Dimsdale (2), with the help of a huge dictionary with natural language words as terms (whereof a small fraction only can be kept in primary storage) and a set of variable-length codes, have reached reductions of up to 75 per cent. By using fixed-length codes for the most frequent pairs of alphabetic characters (digrams), others have compressed their files to 60 per cent of the original size. McCarthy (7) analyses a sample of the data to be compressed and lets the result of the analysis govern the choice of terms on which the Huffman optimum encoding procedure (3) is applied, generating variable-length codes. Using Wagner's technique of dynamic programming (8) to ensure effective encoding, he then records a reduction of 57 per cent in the case of 'some English text'. Huffman's and Wagner's methods are discussed in some detail below.

THE METHOD OF PASTIME

Whatever solution is given to the problem it will certainly have to contain these main features:

1. The entries in the encoding-dictionary must be given, i.e. we must decide on which terms to use.
2. Translations of these entries must be given, i.e. we must decide on which code to use for a term (there must be a one-to-one mapping of the term set on the

code set).

3. A way to use the encoding-dictionary must be given, i.e. we must decide on an encoder that ensures a sufficiently compact coding of the messages.

4. A decoding-dictionary (the inverse of the encoding-dictionary) and a way to use it (a decoder) must be given, i.e. we must decide on a method which economically expands the compressed messages and returns them to their original form.

All four features are readily seen to be interdependent and although some of them or some parts of them have been optimized under a given solution to the rest, we know of no simultaneous optimization of the whole problem. We hold, however, that PASTIME acceptably answers our requirements.

Our second requirement, that PASTIME shall run economically in the 3RIP environment leads us to put certain restrictions on the features 1, 2, and 4. First, we feel that the decoding costs incurred by the Heaps-Dimsdale method (requiring frequent accesses to a decoding-dictionary on secondary storage) are not acceptable and we have therefore decided that the entire decoder and decoding-dictionary must be small enough to be kept in some 150 K bits of primary storage. Thus the number of terms must be restricted to a few thousand. Secondly, it is true that Huffman's optimum encoding method, with a given set of terms and their frequencies in the set of messages to be compressed, produces an optimal set of codes. But the translations of feature 2 will then possibly comprise bitstings of any length up to a value **m**, depending on the number of terms. We feel that a set of codes better adapted to the machine on which the system is to be implemented will make the decoding simpler and cheaper and that this will compensate for the certain loss in reduction. We have therefore decided that the code-lengths should have the form $b(n+i)$ where n is an integral number, $i = 1, 2, 3, 4$, or 5, and b is such that sequences of bitstrings of b bits each easily can be isolated when decoding is performed.

We have described elsewhere (4) how to choose a set of likely term candidates and will therefore only summarily recapitulate the method here:

A sample of the messages on which the final system is to be run is taken and the frequency of every character string, shorter than 16 characters, in the sample is computed. There will of course be an appreciable overlap between the strings but this will have no ill effects for it may obviously be worthwhile to have in the dictionaries not only a given string of a high frequency but also one or more of its substrings; it is equally obvious that if the substring's frequency differs little from that of the superstring then the substring's value in the dictionary is considerably lessened. Therefore a reduced frequency is also computed for each sampled string which is a substring of one or several other sampled strings: the reduced frequency is the frequency of the substring minus the greatest frequency in the set of its superstrings. Should the reduced frequency be zero the substring is excluded. A suitable number of the strings are chosen according to their individual coverage of the sample of messages. The union of the chosen strings and the alphabet of the natural language form the set of term candidates. It should be noted that this is not the definite set of terms. In fact, in testing PASTIME the available parameters controlling the selection process just described were set to yield some 9000 candidates or about three times as many as the expected number of final terms; the less efficient candidates are later weeded out.

The encoding procedure of PASTIME as well as the procedure for selection of the set of terms for the dictionaries make use of Wagner's algorithm which is described below:

Assume a given definite set T of t terms with maximum term-length m, a message in l characters $c_1c_2...c_l$, and a vector $S = (s_1,s_2,...,s_l)$. Wagner's algorithm scans the message from right to left. For each character c_i the algorithm sets s_i = the minimal number of steps necessary to go from c_i to the end of the message. Here a step starting in c_j and ending in

c_{j+k} ($k \leq m$) is defined as identifying the character string $c_j c_{j+1} \ldots c_{j+k}$ as a member of T. To set s_i then means to add one to the smallest of the s-values corresponding to the characters after the last characters of the steps starting in c_i. This involves at most mt+m-1 comparisons and it is seen that the cost of the algorithm which minimizes the number of steps necessary to go through the message is a linear function of the length of the message.

In PASTIME we use a trivial modification of Wagner's algorithm minimizing not the number of steps but the total length of the coded message. Codes of variable lengths $l_1 < l_2 < \ldots < l_j$, the number of codes $n_1, n_2, \ldots, n_j$, respectively, and an ordered sequence of candidate terms are given. The code-length l_1 is given to the n_1 first candidates in the sequence, l_2 to the n_2 next, etc. Then the algorithm encodes a sample of messages, while in the process computing the frequencies of the several codes as used in encoding the sample. The candidates are rearranged in descending order after the frequencies and are fed back to the system with the same sets $L = \{l_i\}$, $N = \{n_i\}$, and the same sample of messages. The procedure is repeated until the difference in cumulated code-length between two different encodings is zero or so small as to be considered of no consequence.

This requires the convergence of the iterative procedure. The requirement is, however, fulfilled, for assume that the reordering of the candidates that constitutes the only difference in the input data for two consecutive encodings meant no more than that a candidate with length l_1 and frequency f_1 in the first encoding changed places in the second with one with code-length l_2 and frequency f_2 in the first. Then it must hold that $(l_1-l_2)(f_1-f_2) \geq 0$. The second optimal encoding cannot produce a result, C_{2opt}, worse than that obtained by the simple change. Let the result of the latter be C and let the result of the first optimal encoding be C_{1opt}. Then we have $C_{1opt}-C_{2opt} \geq C_{1opt}-C = C_{1opt}-(C_{1opt}-l_1f_1-l_2f_2+l_1f_2+l_2f_1) = (l_1-l_2)(f_1-f_2) \geq 0$. The same reasoning holds for any combination of pairs of candidates that have changed places between any two encodings.

The iteration will result in a definite term set consisting of the candidate terms which have actually been used in the last encoding of the iterative procedure. For the given sets L, N, and the initial ordering of the candidate terms, that term set will optimally encode the current sample of messages using the modified Wagner algorithm.

In summary we have decided that

the final dictionary shall contain at most a few thousand terms;

the set of code-lengths L is to be $\{l_i\}$, where $l = b(n+i)$ and $i = 1,2,\ldots,j$;

a term set is to be chosen as follows: first some 9000 candidate terms are selected according to their coverage of a sample of messages, then the modified Wagner algorithm is iteratively applied to the sets L and N, and a sample of messages;

the encoder shall use the modified Wagner algorithm.

It remains to be shown, in testing PASTIME, that

the iterative procedure converges towards roughly the same limit irrespective of the initial ordering of the terms;

the term set works equally well on another sample of messages. This is not self-evident for clearly the very fact that the algorithm is efficiently weeding out candidates less attractive when encoding a first sample of messages will mean that it is highly sensitive to its characteristics. Furthermore the sample used to choose the candidate terms may represent the whole message set badly;

the result of a Huffman-encoding, on a sample of messages, using the term set and the frequencies used and found by a PASTIME-encoding on the same sample, is only marginally better than that of the PASTIME-encoding;

sets L and N yielding good encoding results can be found.

TESTING PASTIME: the tested sample of messages

In order to test PASTIME a testfile BODY has been generated from the bibliographic contents from databases searched within the SDI system at the library of the Royal Institute of Technology in Stockholm. The references are organised in several different ways in the different databases (e.g. there are several methods of denoting author names and journal, volume etc.). The SDI system is not very sensitive to these variations, while it may be expected that PASTIME will behave better on a more homogeneously orgainsed material. All databases have in common, however, that the references are divided into fields containing such information as author, journal, organisation, title, and keywords.

A record in BODY contains a random number, a type and database identifier, and the contents of a field in a reference with every small letter changed to the corresponding capital and every cipher-string changed into the cipher 9. The reason for these changes is that PASTIME is to handle the problems of mixed small and capital letters and of numeric information through the use of shift codes. The file is sorted on the random numbers and the type indicates whether the field from the reference contains what we will call name or text information. We have defined title and keyword fields as text and author, journal, and organisation fields as name information. The reason for this division is that we want to test whether or not two small dictionaries based on different types of messages will yield better results than one bigger based on both types of messages. The database identifier indicates the database the field is selected from. The databases chosen are:

CAC: Chemical Abstracts Condensates
INSPEC: Information Service in Physics, Electrotechnology and Computers & Control
ERIC: Eric Master Files from Educational Resources Information Center
SSC: Social Sciences Citation Index
ISI: Science Citation Index Source Data tape
GRA: Government Reports Announcements

Table 1. The size (in thousands) and the origin of BODY

Type		CAC	INS.	ERIC	SSC	ISI	GRA	TOTAL
Name	no of records	16.5	5.8	3.5				25.8
	no of characters	290	102	74				466
Text	no of records	9.0	6.5	7.2	1.4	2.7	3.3	30.1
	no of characters	300	162	188	82	201	200	1133
Total	no of records	25.6	12.3	10.7	1.4	2.7	3.3	55.9
	no of characters	590	264	262	82	201	200	1599

TESTING PASTIME: the sets of candidate terms

Three basic sets of candidate terms have been used: one for text containing 9000 terms based on all text records in BODY, one for name containing 7000 terms based on all name records in BODY, and one for both name and text containing 9000 terms based on the entire BODY. In all three cases the minimum coverage (the product of the term's frequency and its length) of the relevant part of BODY required for membership in the set has been less than 0.04 per cent so that we may confidently assume that the iterative procedure will give us a term set much smaller than the candidate set. Each of the three basic sets has been ordered after its reduced coverage of the relevant part of BODY when first used in the iterative procedure. By reduced coverage we understand the product of the term's reduced frequency and its length reduced with 12 bits (the 12 bits are intended to represent the likely mean length of the codes). This means that the terms of length one will come last in the ordered sequence of the candidate terms.

TESTING PASTIME: the convergence of the iterative procedure

To test that the initial ordering of the candidates is unimportant we ran the iterative procedure twice on the same sample of 50000 text characters from BODY with L = {9,12,15,18,21} and N = {384,768,1536,3072,3240} and with the basic set of candidates for text alphabetically ordered the second time.

The results obtained were 0.3320 and 0.3332, respectively, which appears to indicate that the initial ordering of the candidate terms is of little significance. The rate of convergence was higher when the normal ordering was used, however.

The procedure converges fairly rapidly. In no case have more than seven cycles of the iteration been necessary for the last two results to differ with less than 0.01 per cent. In a few cases we have allowed the iteration to continue until no more changes between terms were made. This has never taken more than nine cycles.

TESTING PASTIME: different sets L and N

We have said earlier that in the set $L=\{l_i\}$, where l_i is a code-length in bits, l_i should be of the form $b(n+i)$; we have yet to determine b and n and to discuss the set $N = \{n_i\}$. Obviously n_i depends on l_i and we have $n_i \leq 2^{l_i}$. In fact, we can have equality here only if we have exactly one code-length, as we must otherwise signal the length of the code in the code itself. One commonly used method is to flag the length of the code by the number of initial bits all set to one in the code. Let that number be p and let s be an integer such that $1 \leq s \leq b$; it then holds for a code of length l_i that $s(i-1) \leq p \leq si$ except for $l_{max(i)}$ where we have $s(max(i)-1) \leq p \leq l_{max(i)}$.

In table 2: s is indeterminate for test 9 (but then we have only one code-length there); s = 1 for tests 3, 5, and 6; s = 2 for tests 1 and 2; s = 3 for test 4; s = 4 for test 7; and s = 5 for test 8.

A series of tests with the different sets L shown in table 2 were run with the basic candidate terms for text on the same sample of 50000 text characters from BODY. More than one set N was in fact tested for most of the sets L, but in table 2 only the result for the best N for each L is shown. In no case were more than 5200 terms used in the actual encoding (the last cycle of the iteration).

Table 2. 50000 text characters from BODY encoded with the basic candidate terms for text.

Test	L	N	Result
1	9,12,15,18,21	384,768,1536,3072,3240	0.3320
2	8,12,16,20	192,768,3072,4968	0.3418
3	6,9,12,15,18	32,128,512,2048,6280	0.3440
4	9,15,21	448,3584,4968	0.3496
5	4,8,12,16,20	8,64,512,4096,4320	0.3644
6	6,12,18	32,1024,7944	0.3716
7	8,16,24	240,3840,4920	0.3750
8	9,18,27	496,7936,568	0.3781
9	12	4096	0.3939

The (L,N)-pairs of tests 1 and 2 were also used on a material of 400000 text characters from BODY but 1 was still best. No such extensive series was run for name characters but a comparison of the (L,N)-pairs in tests 1, 2, and 7 was made resulting in the same ranking as in table 2.

As the set $L = \{9,12,15,18,21\}$, i.e. $l_i = 3(2+i)$, is the best of those tested, all the following tests use it. Concerning the set N we remain undecided. There is little doubt that if the flagging technique described above is used the set of test 1 in table 2 is the best, but other methods for signalling the code-length could also be considered.

TESTING PASTIME: the number of dictionaries and their size

To enable us to decide on the number of terms a dictionary should contain and also whether separate dictionaries for name and text information would be better than one comprising both types of information, we ran a series of tests, the results of which are displayed in the tables 3a-c. In all the tests 384 codes were of length 9, 768 of length 12, and as many as necessary to reach the total number of terms were of length 15, 18, and 21.

Table 3. Dictionaries based on name, or text, or name and text information from BODY.

Test	BODY-sample	Number of terms	Total length of terms	Result
3a	Name characters:	6940	21243	0.3788
	CAC 200000, ERIC 75000	2926	14235	0.3806
	INS. 100000	2000	8254	0.3908
3b	Text characters:	9000	28394	0.3605
	CAC 100000, SSC 82000,	3000	15473	0.3633
	GRA 100000, INS. 100000	2500	12266	0.3669
		2000	9189	0.3745
3c	Name and text char.	9000	29392	0.3926
	CAC 75000 100000	4000	20197	0.3937
	INS. 50000 75000	3200	15464	0.3940
	ERIC 25000 SSC 70000	2500	11079	0.4003
	GRA 75000	2000	8233	0.4078

As can be seen in table 3, a cut in the size of a dictionary from the basic candidate set down to 3000 terms, while drastically reducing the space needed for the decoding-dictionary, means a loss of reduction not exceeding 0.3 per cent. Further cuts are more expensive, however, and we conclude that a dictionary should contain between 2000 and 3000 terms.

One simple way of storing the decoding-dictionary is to store the terms sequentially and to let their addresses be stored in a vector, whose elements are pointed at by the codes. If two bytes are allowed for the address-elements the storage needed would be twice the number of terms plus the cumulated term-lengths. We judge the space needed for the decoder itself to be negligible.

Then, if no more than 150 K bits are allowed to keep the dictionaries in primary storage, we cannot afford separate dictionaries for name and text information, for if small enough to be stored in 75 K bits each then the reduction would be too small, and, conversely, if the reduction is acceptable then the amount of storage needed would be too large.

It should be noted, however, that in an implementation where a number of databases are run separately through a system, not in parallel as they will in 3RIP, it is undoubtedly advantageous

to construct separate dictionaries for separate databases. We have not been interested in testing this but some indications can be found in table 4. What we have shown here is only that it is not profitable to use more than one dictionary given the restriction on size and the nature of the tested databases. Only when it is known how much space is available for the decoding dictionary and the decoder and what data are to be compressed, should the terms and their number be determined. In table 4 we have restricted ourselves to a dictionary of 2500 terms and have used the set N = {384,748,1348} from the last test but one in table 3c.

Table 4. The dictionary from the last test but one in table 3 used to encode BODY-samples other than that on which the dictionary was formed.

BODY-sample		Result
Name:	CAC 25000	0.4410
Text:	CAC 50000	0.3822
	INS. 25000	0.3856
	ERIC 25000	0.3833
	SSC 12000	0.4227
Name & text	75000	0.4019

It should be kept in mind when looking at table 4 that the dictionary used there yielded the result 0.4003 on the mixed name and text sample on which it was formed. Not unexpectedly, the table shows that the dictionary works better on text than on either of name information only or on name and text information. The relatively high figure for the SSC-sample is explained by the fact that it contains some garbage which should have been excluded, e.g. control information used to facilitate processing in other applications.

The most important result in table 4 is that the reduction is lowered by less than 0.2 per cent when the dictionary is used to encode a mixed sample of information not used in forming the dictionary. This indicates that the sample of 400000 characters used to form the dictionary is representative for the entire file BODY.

TESTING PASTIME: Huffman-coding

We have assigned to the terms of some of our dictionaries the codes Huffman's optimum encoding procedure (3) would give them and have computed the results these Huffman-codes would yield if used to encode the same samples as our dictionaries have been applied to. On the samples on which our dictionaries have been formed, the Huffman-dictionaries have produced reductions between 1 and 1.5 per cent higher than those of our dictionaries based on the set L = {9,12,15,18,21}. Compared to our dictionaries based on the set L = {8,12,16,20,24}, the Huffman-dictionaries have produced between 2 and 2.5 per cent higher reductions. The differences have been higher still for the samples not used to form our dictionaries but they have not exceeded 3 per cent.

It should be borne in mind that in order to assign codes to the terms Huffman's method requires known frequencies for the terms. We have, naturally, used the frequencies given by the modified Wagner-algorithm when encoding the sample. But then again, in order to encode the sample and give the frequencies of the terms as used in the encoding, the modified Wagner-algorithm requires known code-lengths for the codes representing the terms.

Our solution as pointed out above, has been to test the system for a number of pairs of sets L and N but a slight technical modification of PASTIME would allow us to use an iterative procedure applying the modified Wagner-algorithm and Huffman's method in alternation. Such a procedure would seemingly, from a given set of candidate terms and a given sample of messages, form a dictionary which would encode the sample at least as effectively as the dictionaries we have formed from the same candidate terms and the same sample.

THE ENCODER AND THE DECODER; ECONOMIC CONSIDERATIONS

A decoder, operating with such a decoding-dictionary as we have described, will return a code to the corresponding term very fast. When at a given starting point for a code in a

compressed message, it shall:

isolate the code, i.e. find its length from flags in the initial part of the code;

use the code as a pointer to the corresponding element in an address vector;

compute the term-length as the difference between the current element in the address vector and the next element;

use the element as a pointer to the term.

These are all very simple operations and a cycle should not require more than 30 microseconds on the computer to be used for the TRIP system (DEC-10). As the average length of a segment encoded with one code in the tested sample have been between 3 and 4 characters we expect the retrieval time to be less than 10 microseconds per character.

The encoder on the other hand will be more costly but this is acceptable as it will normally be used once only for each message to be compressed. The crucial point is the number of comparisons necessary to identify the terms starting at each character of a message to be encoded. The maximum figure quoted above would in our case with 2500 terms and a maximum term-length of 16 be over 40000. This is indeed inpermissible. However, the number can be drastically reduced by various methods. In testing PASTIME a lookup procedure based on scatter storage of the set of terms was used. Then some 300 microseconds per message character were needed for the encoding procedure when executed on an IBM 360/75 computer. For the production version of the system a prefix tree structure, devised by Löfström (6), taking advantage of the characteristics of the distribution of the terms in the messages to be encoded will be used. We estimate that the total encoding time per character then will be at most 150 microseconds.

SUMMARY

We aim towards a system which will compress reasonably homogeneous large textual files (e.g. consisting of bibliographic references) to about 40 per cent of their original size by uniquely translating variable-length character strings into

variable-length codes. The code-lengths shall be suited to rapid processing in the computer on which the system is to be implemented (e.g. 9, 12, 15, ... on a DEC-10 or 8, 12, 16,... on an IBM computer). Returning of the compressed items to their original form must be fast and cheap and to that end we require the decoder and decoding-dictionary to be small on the order of 150 K bits.

We assume, then, that:

1. the file F is to be compressed;
2. the decoder and the decoding-dictionary shall occupy no more than S bits;
3. a set L of code-lengths is given; and
4. a set N giving the number of codes for each code-length in L is given.

We then:

1. take a representative sample f of F (approximately half a million characters);
2. let PASTIME choose some 9000 candidate terms according to their coverage of f; these candidates form the set T;
3. apply the modified Wagner-algorithm to L, N, T, and f giving us a set of terms T_1 optimally encoding f for the given L, N, and T.

If T_1 is such that the corresponding decoder and decoding-dictionary require less than S bits all is well. Otherwise the sets L and N must be modified to L_1, N_1 in order to make T_1 smaller and step 3 must be run with the new indata L_1, N_1, T_1 and f giving a new term set T_2. Iteration of this procedure will give us the definite constellation L_d, N_d, and T_d. The iteration should be continued until an acceptable compromise has been reached between the yielded compression result and the space requirements for the corresponding decoder and decoding-dictionary. Our test imply that no serious conflicts arise until the number of terms is reduced to less than 3000.

Using the method given we can now construct an encoder and a decoder compressing the textual fields of F to no more than

40 per cent of their original size. When run on either of the large computers of Stockholms datamaskincentral the costs associated with encoding and decoding one million characters are estimated to fall below 80 and 5 Swedish crowns, respectively.

REFERENCES

1) H.S. Heaps: Storage Analysis of a Compression Coding for Document Data Bases {*INFOR 10*(1972)pp 47-61}

2) H.S. Heaps and J.J. Dimsdale: File Structure for an On-line Catalog of One Million Titles {*Journal of Library Automation 6*(1973)37}

3) D.A. Huffman: A Method for Construction of Minimum Redunduncy Codes {*Proceedings of the the IRE 40*(1952)pp 1098-1101}

4) J. Hultgren and R. Larsson: *A Method Defining a Limited Set of Character-strings with Maximal Coverage of a Sample of Text* {TRITA-LIB-4020a, June 1973}

5) R. Larsson et al.: *3RIP: File Design for the Search System* {TRITA-LIB-4042, March 1975}

6) M. Löfström: *On the Construction of a Prefix Tree when the Access Frequency Distribution is Non-uniform* {University of Sheffield, Master Thesis 1973}

7) J.P. McCarthy: Automatic File Compression {*International Computing Symposium 1973* (1974) pp. 511-516}